Université catholique de Louvain
Faculté des Bioingénieurs
Louvain-la-Neuve

Questions spéciales de gestion de l'information

LBRTI2202

Détection d'oiseaux par réseaux de neurones

Auteurs :
Guillaume De Meue
Ludovic Lepers
Maxime Messens
Henri Moens de Hase
Dilhan Ozturk

Encadrants :
Pr. Patrick Bogaert
Pr. Emmanuel Hanert

Année académique
2019-2020

Table des matières

1 Introduction **1**

2 Matériel et méthodes **1**
2.1 Présentation des données 1
2.2 Prétraitement des fichiers audio 2
2.3 Chargement des données et partitions pour l'entraînement et la validation 4
2.4 Description du réseau 4
2.5 Paramètres d'entraînement du modèle 7
2.6 Outils d'évaluation 8

3 Résultats et discussion **8**
3.1 Entraînement d'un réseau de 4 couches, pour 45 epochs 8
3.2 Entraînement d'un réseau de 3 couches, pour 45 epochs 11
3.3 Entraînement d'un réseau de 3 couches, pour 25 epochs 13

4 Conclusion **15**

Références **16**

5 Annexes **18**
5.1 Fonction d'activation de type ReLU 18
5.2 Pooling 18

1 Introduction

Le machine learning et l'apprentissage supervisé par réseaux de neurones montrent depuis quelques années leurs capacités à détecter les informations utiles au sein d'un signal pour faire de la reconnaissance d'images avec un taux élevé de précision. Plus particulièrement, les réseaux de neurones ont connu un grand succès dans de nombreuses tâches de traitement d'images. Ils ont notamment permis de créer des systèmes de classification d'images très performants [Giacinto and Roli, 2001], ainsi que d'améliorer l'objectivité et l'efficacité des diagnostics histopathologiques sur base d'images de biopsie [Litjens et al., 2016].

De plus, ces réseaux ont aussi montré leurs capacités pour la reconnaissance de sons et beaucoup de travaux se sont portés sur la classification de sons urbains [Piczak, 2015]. Cependant, l'utilisation d'un son pour entraîner un réseau de neurones est fort complexe. Afin de reconnaître les sons environnants, ces réseaux utilisent dans la majorité des cas des spectres de fréquences pour représenter le son sous forme d'image [Li et al., 2017].

En 2016, la Queen Mary University of London en collaboration avec la IEEE Signal Processing Society a lancé un concours pour la création d'un algorithme de détection d'oiseaux [Stowell et al., 2016]. Le but est de concevoir un système qui mentionne soit la présence, soit l'absence de sons produits par des oiseaux dans un court enregistrement audio. La détection de sons d'oiseaux dans des enregistrements audio peut trouver des applications notamment dans la surveillance automatique de la faune ou dans la gestion de bibliothèques audio. Pour concevoir l'algorithme, des jeux de données sont mis à disposition des participants. Ceux-ci consistent en de courts extraits sonores enregistrés en différents lieux et types d'environnement.

Afin de renforcer nos compétences dans le domaine du machine learning, nous avons décidé d'implémenter un réseau de neurones pour relever le défi de la Queen Mary University of London. Pour parvenir à déterminer la présence ou non de sons d'oiseaux dans des enregistrements audio, nous avons transformé les fichiers sonores fournis en images représentant leurs spectres de fréquences afin d'utiliser un réseau de convolution adapté à la reconnaissance d'images. Ce réseau a été implémenté en utilisant la librairie Keras [Chollet, 2017] de Python qui a produit des résultats probants pour d'autres réseaux de neurones. Deux réseaux de neurones ont été utilisés et différentes méthodes de prétraitement ont été testées pour améliorer l'efficacité du réseau de neurones.

2 Matériel et méthodes

2.1 Présentation des données

Jeu de données 1 : freefield1010

Ce jeu de données contient 7 690 enregistrements effectués à travers le monde et collectés par le projet *FreeSound* [Stowell and Plumbley, 2013] dans différents types d'environnement. Ce sont des fichiers **.**wav mono en qualité CD standard (16-bits, 44,1 kHz), durant une dizaine de secondes et normalisés en amplitude. Un fichier **.**csv s'y rapportant permet de labelliser le jeu de données. Il reprend sur chaque ligne le nom du fichier audio concerné et un chiffre : 0 pour l'absence d'oiseaux et 1 pour la présence d'oiseaux. Il contient 5 755 fichiers sans oiseaux (75%) et 1 935 avec oiseaux (25%).

Jeu de données 2 : Warblr

Ce jeu de données contient 8 000 enregistrements obtenus par une production participative au Royaume-Uni. Ces données ont été enregistrées dans de nombreux endroits et environnements. On y retrouve des bruits de trafic, des conversations, des imitations d'oiseaux, etc. Il s'agit d'extraits sonores de 10 secondes qui sont labellisés de la même façon que le premier jeu de données. Il contient 1 955 fichiers sans oiseaux (24%) et 6 045 fichiers avec oiseaux (76%).

Jeu de données 1 + Jeu de données 2

Au total, on dispose donc de 15 690 extraits sonores parmi lesquels 7 710 sans oiseaux (49%) et 7 980 avec oiseaux (51%), correspondant à un jeu de données bien équilibré.

Les deux jeux de données sont téléchargeables séparément sur la page web du concours [Stowell et al., 2016].

2.2 Prétraitement des fichiers audio

Les réseaux de neurones de classification par apprentissage supervisé sont connus et bien documentés, en particulier pour la reconnaissance d'images. Une technique utilisée pour analyser des fichiers audio consiste à transformer l'information contenue dans le son en une image afin de pouvoir utiliser des archétypes connus de réseaux de classification.

L'opération choisie est l'utilisation du Mel-Frequency Cepstral Coefficients (MFCC). Le MFCC est une représentation spectrale du son, fort utilisée pour l'analyse de discours car elle tente de prendre en compte l'effet de filtre que le système vocal humain induit sur le son qu'il génère. Nous avons choisi ce système de prétraitement car les algorithmes de transformation disponibles nous permettaient de travailler facilement sur nos données. D'autres représentations spectrales sont possibles telles que le log Mel-spectrum mais l'efficience d'une représentation spectrale dépend également du modèle du réseau de neurones qui l'utilise, ce qui rend le choix de la représentation spectrale difficile [Li et al., 2017]. Toutefois, le MFCC est une valeur sûre qui a fait ses preuves dans différents concours de machine learning en 2013 et en 2016 (Ibid.).

Une fois les fichiers audio au format **.**wav transformés en MFCC (à l'aide de la librairie LibROSA [McFee et al., 2020]), nous les avons enregistrés au format **.**npy afin de permettre leur réutilisation dans Python sans avoir à recommencer la transformation à chaque utilisation de ceux-ci. A titre d'exemple, la Figure 1 affiche un fichier audio transformé en MFCC.

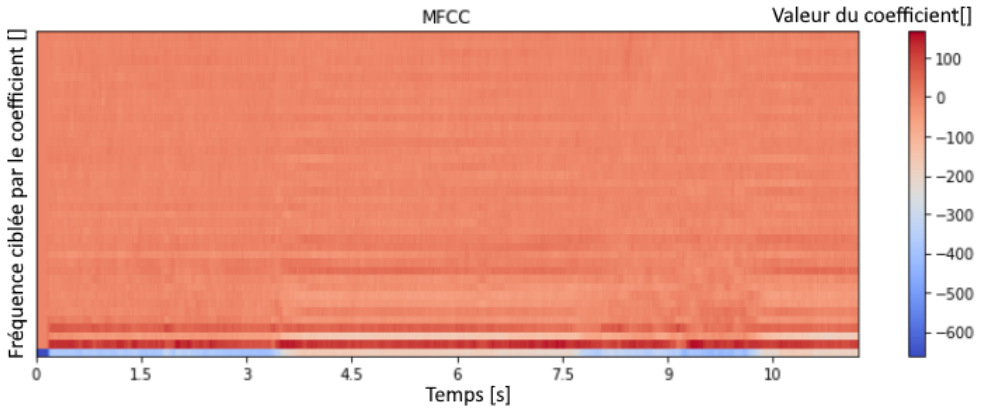

FIGURE 1 – Fichier transformé en format MFCC. Chaque colonne représente un intervalle de temps selon l'axe horizontal. Chaque ligne représente la "fréquence" concernée par le coefficient. La couleur de chaque point représente la valeur du coefficient.

Données "filtrées"

Nous avons cherché à éliminer une partie du bruit qui pourrait être présent dans les fichiers originaux dans le but de fournir une information plus claire au réseau de neurones. En supposant que les chants d'oiseaux se situent plutôt dans les hautes fréquences, nous avons retiré la ligne la plus basse du MFCC, à savoir celle correspondant aux plus basses fréquences (en fixant tous les coefficients qui la composent à 0). Sur la Figure 1, cela correspond à la ligne partiellement bleue qui possède par moment des coefficients fortement négatifs. Nous nous sommes limité à la première ligne par crainte de supprimer trop d'information. Il aurait été intéressant d'essayer de fixer les coefficients des quelques premières lignes à 0 mais cela n'a pas été fait dans ce travail. N'ayant aucune garantie sur l'efficacité de cette méthode, les fichiers MFCC initiaux sont conservés et le réseau de neurones est testé aussi bien avec les MFCC initiaux qu'avec les "filtrés" (nous utilisons le terme "filtré" car la transformation simule l'utilisation d'un filtre passe-haut sur le fichier audio). Étant donné que nous utilisons du "Max Pooling", mettre à zéro une ligne revient en théorie à supprimer cette ligne. En effet, le "Max Pooling" est une méthode de mise en commun par le maximum, pour laquelle seul le neurone avec la valeur de sortie la plus élevée par rapport à ses voisins est conservé entre deux couches de convolution (voir l'Annexe 5.2).

La Figure 2 illustre le résultat de ce filtre appliqué sur l'exemple de la Figure 1. Les couleurs sont différentes par rapport à la Figure 1 à cause de l'adaptation de l'échelle à la nouvelle gamme de valeurs. En effet, la ligne mise à zéro contient les coefficients les plus élevés en valeur absolue.

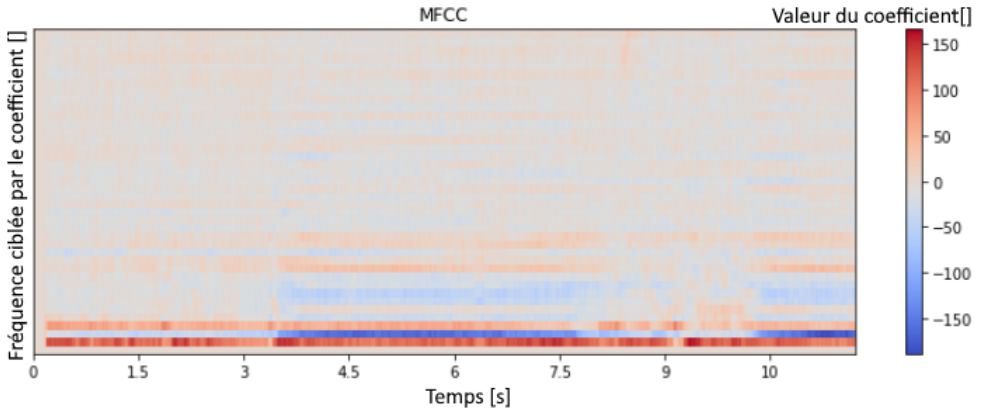

FIGURE 2 – Exemple d'un MFCC "filtré" où la valeur de la dernière ligne correspondant à la plus basse fréquence est nulle en tout temps

2.3 Chargement des données et partitions pour l'entraînement et la validation

Afin d'utiliser l'ensemble des données pour l'entraînement, nous avons créé un générateur de batch. Le batch fait le lien entre l'algorithme d'apprentissage et le disque dur de l'ordinateur, en fournissant à l'algorithme les données d'apprentissage uniquement lorsque ce dernier en a besoin. Ceci permet d'utiliser l'ensemble des données disponibles sans avoir à les stocker toutes simultanément dans la mémoire vive.

Dans la mesure où nous sommes capables de charger suffisamment de données, nous pouvons travailler avec les 2 jeux de données regroupés. Cependant, afin de pouvoir évaluer les performances de notre réseau de neurones, il est important de conserver une partie des données qui ne peut être utilisée pour l'entraînement. En effet, nous devons être en mesure de confronter le modèle à des données qu'il n'a pas encore rencontrées. Pour ce faire, nous commençons par regrouper les deux jeux de données sans faire de distinction entre les fichiers selon leur origine. Les fichiers `.csv` contenant les labels des images ont également été fusionnés. Ensuite, nous utilisons la fonction `train_test_split` de la librairie scikit-learn [Pedregosa et al., 2011] qui permet de créer deux sous-ensembles de manière aléatoire : un pour l'apprentissage et l'autre pour la validation. La fonction `train_test_split` a été utilisée avec ses paramètres par défaut, conduisant à la formation de partitions contenant 75% des données pour l'entraînement et 25% pour la validation. Cette séparation des données est faite avant le début de chaque entraînement du réseau. Les données sur lesquelles sont entraînés les réseaux ne sont donc pas les mêmes à chaque entraînement. L'échantillonnage s'étant fait de manière aléatoire sur un nombre de données important, nous supposons que les différents choix de partitions n'impactent pas significativement les résultats des modèles.

2.4 Description du réseau

Deux réseaux de neurones différents ont été utilisés dans le cadre de ce projet. Ils sont tirés du travail de Mike Smales, qui a effectué de la reconnaissance de sons sur des fichiers MFCC [Smales, 2019].

Le premier réseau de neurones utilisé est presque identique à celui utilisé par Mike Smales. Il s'agit d'un réseau composé de quatre couches de convolution dont chacune est composée d'une fenêtre de convolution de taille 2 et d'une fonction de max pooling, également de taille 2 (chaque neurone conservant la valeur la plus élevée des quatre neurones auxquels il est connecté, voir

l'Annexe 5.2). La fonction d'activation des couches de convolution est de type ReLU (Rectified Linear Unit) [Dahl et al., 2013] (voir l'Annexe 5.1). Le nombre de filtres par couche est multiplié par un facteur 2 entre chaque couche, en commençant à la valeur 16 pour la première couche et à la valeur 128 pour la dernière couche. Enfin, après une couche d'average pooling global (une méthode de mise en commun similaire au max pooling où l'on fait la moyenne des valeurs de patch d'entrée, voir l'Annexe 5.2), une couche de sortie composée de deux neurones donne les résultats de la classification. Ils suivent une fonction d'activation "softmax", qui permet d'obtenir en sortie un vecteur de probabilités d'appartenance aux classes. Les deux neurones de sortie fournissent une valeur entre 0 et 1, avec la valeur de sortie d'un neurone égale à 1 moins la valeur de sortie de l'autre neurone. Cette valeur sera interprétée comme la "certitude" du modèle. Mike Smales utilisait plus de neurones de sortie dans son modèle pour permettre de classer des données dans plus de deux catégories.

Il est à noter qu'il existe des couches dites de "dropout" dans notre modèle. L'introduction de ces couches sert à éviter le *surajustement*[1] dans les réseaux de neurones. L'idée consiste à mettre à zéro la sortie de neurones sélectionnés au hasard pendant l'entraînement pour éviter ainsi de la "co-adaptation" [Srivastava et al., 2014]. Cela force chaque unité à apprendre correctement indépendamment des autres. Dans notre cas, la sortie de 20% des neurones est mise à zéro aléatoirement. Les couches de dropout peuvent également permettre une accélération de l'apprentissage.

Le deuxième réseau utilisé est le même que le précédent sauf qu'il n'est composé que de trois couches de convolution. Nous avons décidé de retirer une couche car malgré un nombre relativement important de données, nous avons estimé qu'il restait possible d'avoir un problème de *surajustement* du modèle (le nombre de paramètres du modèle étant supérieur au nombre de données). Concrètement, il s'agit simplement de supprimer la dernière couche de convolution, de max pooling et de dropout. La couche d'average pooling global est alors un vecteur de 64 valeurs (et non de 128 valeurs).

1. L'overfitting ou le surajustement dans le cas des réseaux neuronaux est un phénomène où un modèle apprend à classer les données auxquelles il a été confrontées sans réussir à identifier les informations pertinentes pour effectuer la classification. Un modèle surajusté aura un taux de prédiction excellent pour les données sur lesquelles il a effectué son apprentissage mais ne sera pas aussi performant pour classer de nouvelles données.

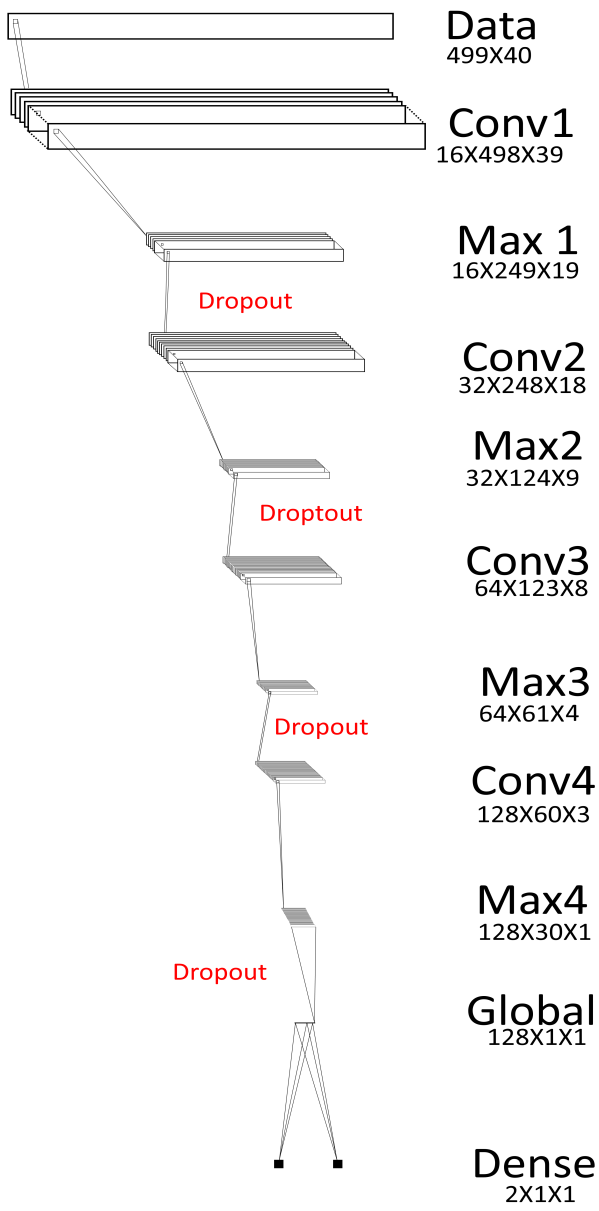

FIGURE 3 – Schéma de l'agencement des différentes couches du modèle à quatre couches de convolution. La position des couches de dropout est représentée en rouge. *Data* correspond à la donnée entrée, *Conv* aux couches de convolution, *Max* aux couches de max pooling, *Global* à la couche d'average pooling global et *Dense* à la couche composée des deux neurones finaux. Les couches de max pooling divisent par quatre la taille des matrices et les couches de convolution retirent une ligne et une colonne car les valeurs au bord ne sont visitées que deux fois au lieu de quatre.

Le modèle présenté à la Figure 3 contient 43 570 paramètres répartis dans les couches de convolution et la couche finale. Le nombre de paramètres par couche est présenté dans le tableau suivant.

Couche	Nombre de filtres	Taille des filtres	Total
Convolution 1	16	4(2X2)	80
Convolution 2	32	4	2080
Convolution 3	64	4	8256
Convolution 4	128	4	32896
Final	2 (neurones)	128(input/neurone)	258

Le nombre de paramètres dans les couches de convolution est donné par

$$\text{nombre de paramètres} = n_C \times t \times n_{C-1} + n_C$$

avec

n_C le nombre de filtres dans la couche,

n_{C-1} le nombre de filtres dans la couche précédente (n_{C-1} vaut 1 pour la première couche),

t la taille des filtres.

Pour la couche finale, il y a 128 entrées avec un biais pour chaque neurone, ce qui donne 258 paramètres (128 +1 pour chaque neurone). Le biais est une valeur entrée dans le neurone, toujours égale à 1, et comme toutes les autres valeurs entrées, elle est multipliée par un paramètre. Le modèle avec trois couches de convolution possède 10 546 paramètres, la différence venant de la quatrième couche de convolution qui est simplement absente et des deux neurones de sortie, qui ont chacun 64 entrées et un biais, ce qui fait 130 paramètres pour la couche.

2.5 Paramètres d'entraînement du modèle

La fonction `model.compile` de Keras [Chollet, 2017] nous permet de configurer le modèle pour l'entraînement. Nous devons lui fournir un *optimizer* et une fonction de perte. L'*optimizer* choisi est *adam* (pour adaptative moment estimation). Il se base sur la descente du gradient stochastique mais avec des taux d'apprentissage variables [Brownlee, 2017]. Il requiert relativement peu de mémoire et donne généralement de meilleurs résultats que les autres techniques adaptatives [Zhang, 2018]. Pour la fonction de perte, nous avons choisi la *categorical_ crossentropy*. Cette fonction de perte évalue la performance du modèle en tenant compte de la certitude des prédictions. Elle est ici équivalente à la perte logarithmique. Pour une observation, elle est donnée par

$$\text{perte logarithmique} = -(y \times \log(p) + (1 - y) \times \log(1 - p))$$

avec

y valant 0 ou 1 selon la classe de l'observation,

p la probabilité prédite par le modèle d'appartenir à la classe en question.

Sa valeur augmente quand la probabilité prédite s'écarte du label observé. Une mauvaise prédiction avec un haut niveau de confiance est fortement pénalisée. Le modèle s'entraîne donc en essayant de minimiser cette valeur [Nielsen, 2019].

L'entraînement en tant que tel est réalisé à l'aide de la fonction `model.fit_generator` de Keras [Chollet, 2017]. Nous lui précisons les données d'entraînement et de validation, ainsi qu'un nombre d'epochs. Ce nombre d'epochs correspond au nombre de fois que l'ensemble des données d'entraînement vont être parcourues. On s'attend à ce que la fonction de perte diminue au fil des epochs.

2.6 Outils d'évaluation

Pour ce qui est de l'évaluation, nous avons choisi différents outils qui permettent de comparer les différents réseaux.

La **matrice de confusion** met en relation les prédictions du modèle sur le jeu de validation avec la réalité correspondant aux labels.

Prédiction / Réalité	absence	présence
absence	Vrais négatifs	Faux positifs
présence	Faux négatifs	Vrais positifs

La réalité est déterminée sur base du label (0 pour absence, 1 pour présence). Pour la prédiction, rappelons que nous avons deux neurones de sortie, un donnant la probabilité de présence et l'autre donnant la probabilité d'absence. La plus grande de ces deux valeurs détermine le choix de l'absence ou de la présence d'oiseaux.

Cette matrice permet de calculer différents indicateurs tels que l'*accuracy*. L'*accuracy* est souvent utilisée et correspond à la proportion d'éléments correctement classés, c'est-à-dire situés sur la première diagonale. On peut également facilement déduire la proportion d'éléments correctement classés au sein d'une classe. Dans le but de nous faire une idée de la façon dont évolue la qualité du modèle au fil de l'entraînement, nous pouvons préciser l'*accuracy* comme métrique d'évaluation dans la fonction `model_compile` afin de la représenter en fonction du nombre d'epochs parcourues. On s'attend à ce qu'elle augmente et tende vers un plateau.

L'*accuracy* et la matrice de confusion sont assez simples conceptuellement mais ne permettent pas de prendre en compte la "certitude" de la prédiction. En effet, une prédiction de présence d'oiseaux avec une "certitude" de 51% ou de 99% sont classées de la même façon, puisque les deux valeurs seront associées à la valeur 1 (présence). Des histogrammes représentant pour chacune des classes le nombre de prédictions en fonction de leur niveau de confiance seront utilisés pour nous faire une idée plus précise de cette "certitude".

3 Résultats et discussion

3.1 Entraînement d'un réseau de 4 couches, pour 45 epochs

Avec les données filtrées

Avant d'entraîner un réseau, il faut définir le nombre d'epochs qui vont être effectuées. Il semble évident que plus un modèle passe par un nombre important d'epochs, plus il sera à même de classer des données correctement. Il y a toutefois un problème : étant donné que le modèle est constitué de plus de 43 000 paramètres dans sa version la plus lourde, et qu'il n'y a qu'environ 15 000 données (moins les données de validation), il y a un risque de *surajustement*.

Nous commençons par présenter les résultats de l'entraînement d'un réseau de 4 couches, entraîné sur les données filtrées.

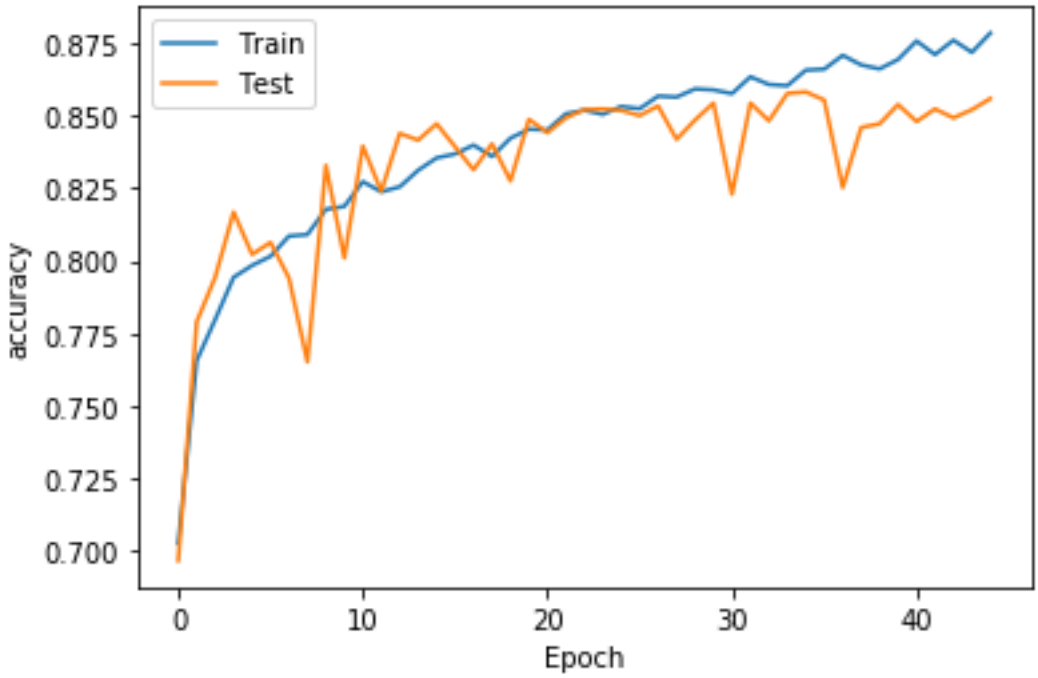

FIGURE 4 – Évolution de l'*accuracy* du modèle en fonction du nombre d'epochs pour les données d'entraînement (Train) et de validation (Test). Au-delà de 25 epochs, l'*accuracy* des données de validation plafonne, alors que celle des données d'entraînement continue d'augmenter.

Comme visible sur la Figure 4, l'*accuracy* augmente dans un premier temps, aussi bien pour les données d'entraînement que pour les données de validation. Conformément aux attentes, au plus les données sont parcourues un grand nombre de fois, au plus les prédictions du modèle s'améliorent. Toutefois, à partir d'environ 25 epochs, l'*accuracy* des données d'entraînement continue d'augmenter alors que l'*accuracy* des données de validation plafonne. Cela signifie que le modèle ne devient meilleur que pour reconnaître les données d'entraînement mais pas pour reconnaître de nouvelles données. On considère alors qu'il y a *surajustement*.

Pour déterminer si le modèle a de bonnes performances sur nos données de validation, on peut regarder la matrice de confusion.

Prédiction Réalité	absence	présence
absence	1794	120
présence	343	1658

On observe que les données correctement classées sont majoritaires. L'*accuracy* vaut 88%. De plus, la classification s'effectue correctement au sein de chacune des deux classes. En effet, 83% des extraits avec oiseaux sont identifiés comme tels et 94% des extraits sans oiseaux sont bien reconnus.

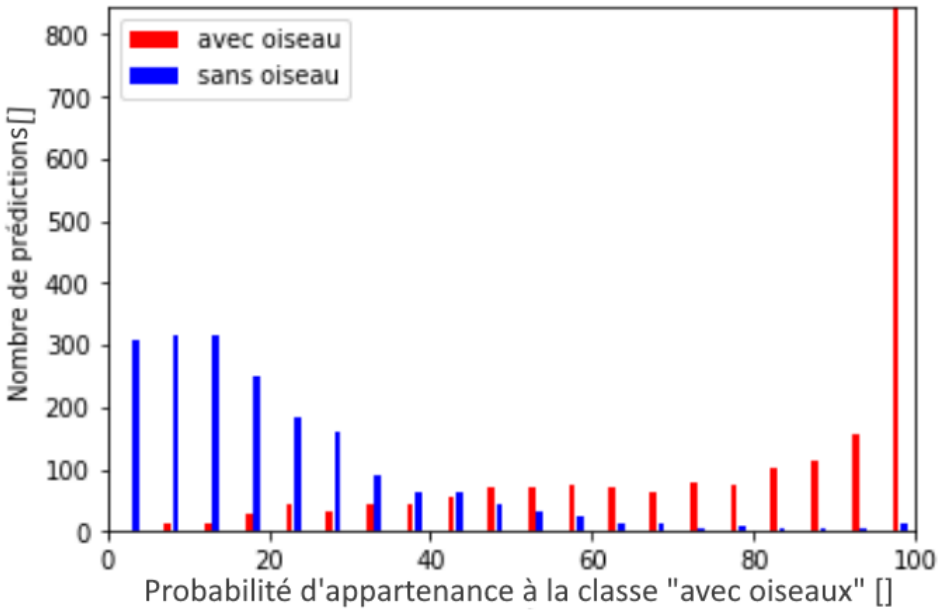

FIGURE 5 – Histogramme du nombre de prédictions en fonction de leur niveau de confiance pour chacune des classes d'un modèle à 4 couches durant 45 epochs sur les données filtrées

La Figure 5 présente la distribution du nombre de prédictions en fonction de la probabilité d'appartenance à la classe "avec oiseaux". Pour chaque extrait sonore, le modèle calcule une valeur de prédiction comprise entre 0 et 1. Une prédiction égale à 0 correspond à l'absence d'oiseaux avec une grande certitude et une prédiction égale à 1 signifie la présence d'oiseaux avec une grande certitude. Ces valeurs ont été rassemblées par intervalle de 0,05 (l'axe des abscisses a été exprimé en pourcents pour plus de clarté). Si les barres (rouges ou bleues) sont grandes, cela signifie qu'il y a eu beaucoup de prédictions pour cette probabilité d'appartenance.

La distribution rouge correspond aux prédictions des extraits sonores avec chant d'oiseaux, tandis que la bleue correspond aux prédictions des extraits sonores sans oiseaux. Une situation idéale correspond à deux distributions concentrées aux extrémités. Ainsi, le modèle ne se tromperait pas et aurait beaucoup de certitude sur ses prédictions. A l'opposé, si les deux distributions sont au milieu, le modèle ne serait pas meilleur qu'un modèle probabiliste aléatoire.

Avec les données non filtrées

Afin d'évaluer l'effet du filtrage des données, il est intéressant de comparer les résultats précédents aux résultats obtenus suite à l'entraînement du même réseau de 4 couches, avec des données non-filtrées. La matrice de confusion pour les données non-filtrées est présentée ci-dessous.

Prédiction / Réalité	absence	présence
absence	1513	357
présence	270	1775

La plupart des prédictions sont toujours correctes et ce pour les deux classes. Le taux de reconnaissance des extraits avec oiseaux a légèrement augmenté, atteignant 87%. Cependant, seuls 81% des extraits sans oiseaux sont prédits correctement. Ces derniers semblent donc

moins bien identifiés que précédemment. Cela se répercute sur l'*accuracy*, légèrement en baisse et valant 84%.

Il n'y a cependant pas une différence importante avec les données filtrées. Il en va de même lorsqu'on regarde l'histogramme à la Figure 6, assez similaire à celui de la Figure 5. A ce stade, nous ne privilégions pas encore l'usage des données filtrées par rapport aux données non-filtrées.

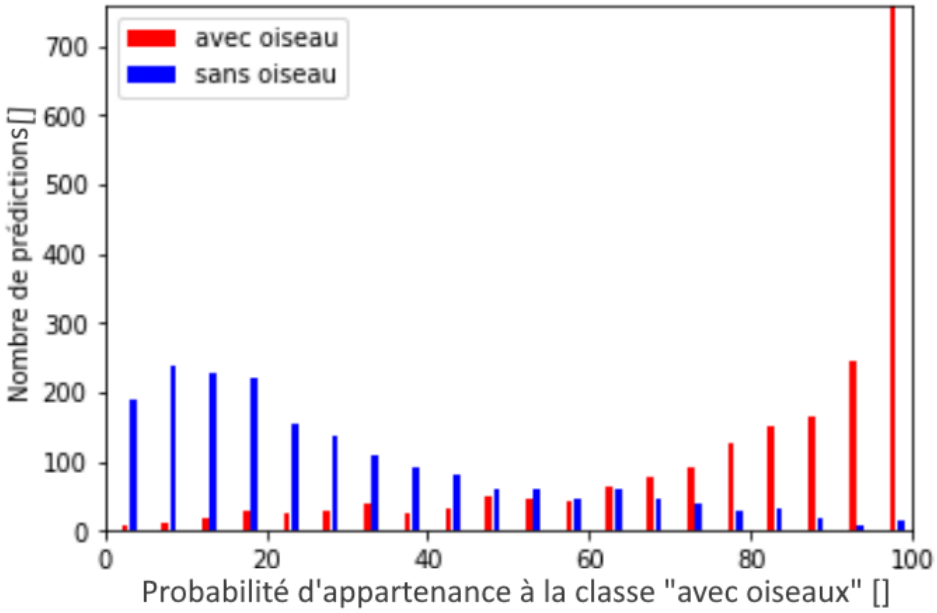

FIGURE 6 – Histogramme du nombre de prédictions en fonction de leur niveau de confiance pour chacune des classes d'un modèle à 4 couches durant 45 epochs sur les données non filtrées

3.2 Entraînement d'un réseau de 3 couches, pour 45 epochs

Avec les données filtrées

Comme évoqué précédemment, le nombre de paramètres du réseau de neurones à 4 couches est largement supérieur au nombre d'observations dont nous disposons. En enlevant la dernière couche, le nombre de paramètres baisse sensiblement, passant de 43 570 à 10 546.

La matrice de confusion ci-dessous donne les résultats de la classification obtenus après l'entraînement du réseau de neurones à trois couches avec les données filtrées.

Prédiction / Réalité	absence	présence
absence	1642	272
présence	215	1786

On obtient une *accuracy* de 88%, équivalent à celui obtenu pour le réseau de neurones à 4 couches. La suppression de la dernière couche ne semble donc pas nuire à l'*accuracy*. Parmi les extraits avec oiseaux, 89% sont identifiés correctement. Pour les extraits sans oiseaux, 86% sont bien reconnus. Ici, on semble avoir une meilleure confiance dans la détection des oiseaux, ce qui est bien traduit par le diagramme de la Figure 7 dans lequel les données à l'extrême droite sont plus nombreuses. En contrepartie, les données sans oiseaux sont un peu plus au centre.

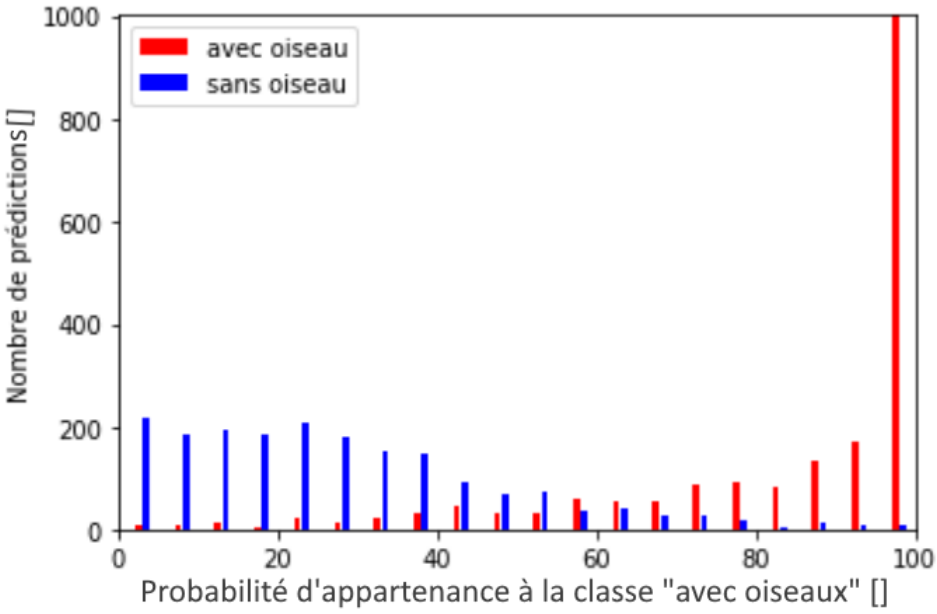

FIGURE 7 – Histogramme du nombre de prédictions en fonction de leur niveau de confiance pour chacune des classes d'un modèle de 3 couches durant 45 epochs sur les données filtrées

Avec les données non-filtrées

Afin de poursuivre l'investigation de l'effet du filtre, le réseau de neurones de 3 couches est également entraîné avec les données non-filtrées. La matrice de confusion associée est présentée ci-dessous.

Prédiction / Réalité	absence	présence
absence	1651	219
présence	402	1643

Contrairement à ce qu'on a pu voir avec le réseau de 4 couches, c'est cette fois l'identification des extraits avec oiseaux qui semble baisser légèrement (80% de prédictions correctes), en comparaison avec l'usage des données filtrées. On ne déduira donc pas que le filtre nuit à la détection des extraits avec oiseaux. L'application du filtre semblait améliorer la reconnaissance des extraits sans oiseaux dans le réseau à 4 couches. Cela ne semble pas être le cas ici, puisque pour cette classe, on obtient un taux de reconnaissance de 88% très proche de ce qu'on obtient avec les données filtrées. Néanmoins, tout comme pour le réseau à 4 couches, on observe une légère baisse de l'*accuracy* (valant ici 84%), ce qui encourage l'usage des données filtrées. De plus, si les résultats binaires de la classification restent assez bons, on observe sur le diagramme de la Figure 8 des valeurs de prédiction situées plus au centre, diminuant ainsi le nombre de données avec une probabilité très élevée d'être correctement classées.

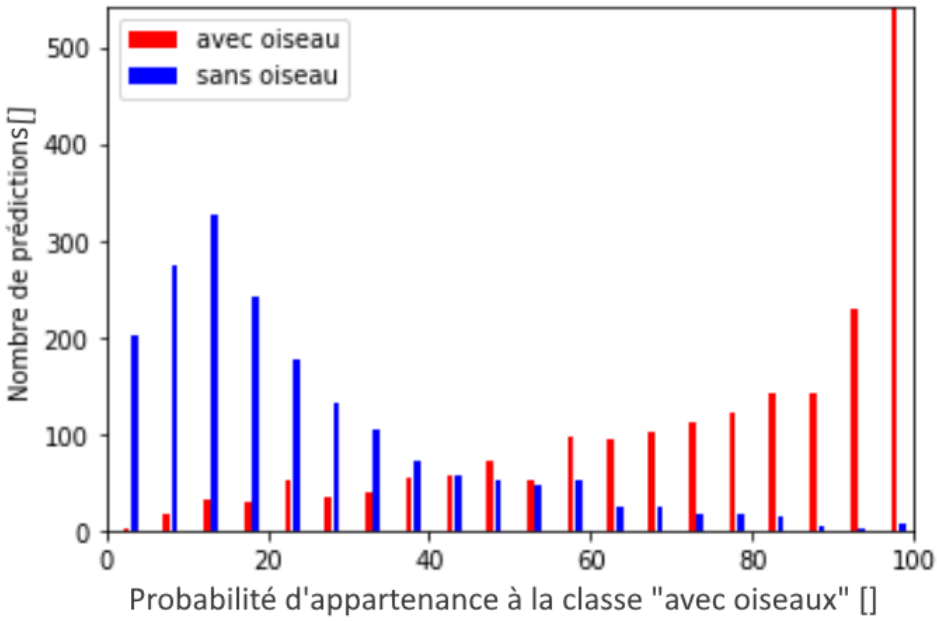

FIGURE 8 – Histogramme du nombre de prédictions en fonction de leur niveau de confiance pour chacune des classes d'un modèle de 3 couches durant 45 epochs sur les données non filtrées

3.3 Entraînement d'un réseau de 3 couches, pour 25 epochs

Comme nous l'avons vu à la Figure 4, entraîner le modèle pour 45 epochs semble un peu long et peut mener à un *surajustement*. En utilisant le réseau à 3 couches sur les données filtrées, nous avons essayé de diminuer ce nombre d'epochs afin de réduire le temps d'entraînement. Nous avons choisi 25 epochs car c'est à partir de ce nombre qu'on observe la dissociation entre les courbes d'*accuracy* des données d'entraînement et des données de validation.

Avec les données filtrées

Nous avons vu dans les résultats précédents que le filtrage des données a amélioré nos scores d'*accuracy*. Nous ne présentons donc ci-dessous que les résultats de l'entraînement du réseau sur les données filtrées.

La matrice de confusion ci-dessous nous indique que la classification s'effectue déjà correctement après 25 epochs.

Prédiction / Réalité	absence	présence
absence	1639	275
présence	389	1612

Malgré une forte diminution du nombre d'epochs, nous gardons plus de 80% de données correctement attribuées pour chacune des classes : 81% pour les extraits avec oiseaux et 86% pour les extraits sans oiseaux. On remarque malgré tout une baisse de l'*accuracy* (valant dans ce cas 83%) et la Figure 9 traduit des distributions moins distinctes qu'auparavant entre les deux classes.

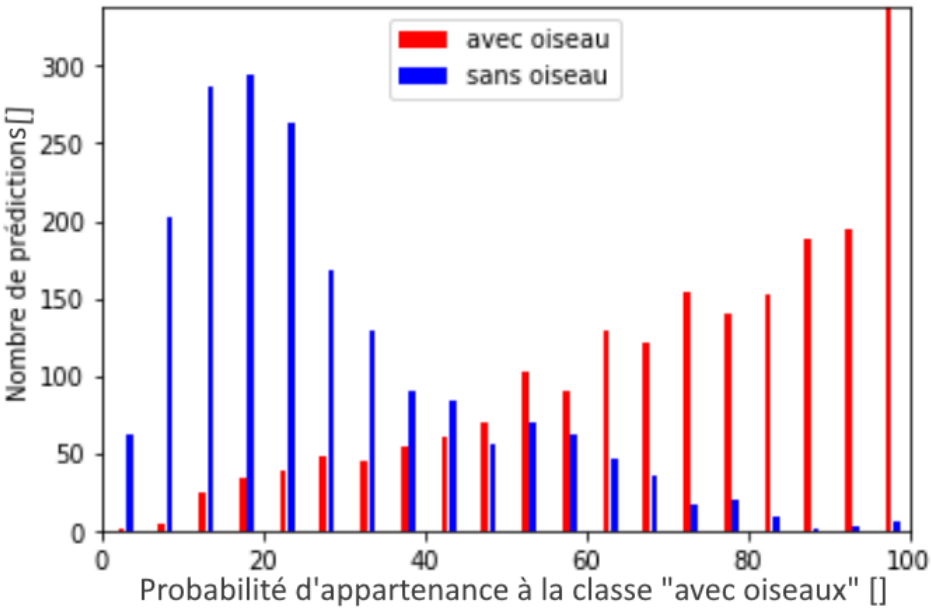

FIGURE 9 – Histogramme du nombre de prédictions en fonction de leur niveau de confiance pour la présence d'oiseaux d'un modèle de 3 couches durant 25 epochs sur les données filtrées

4 Conclusion

L'utilisation de l'apprentissage supervisé par réseaux de neurones pour la détection de la faune ornithologique semble prometteuse. Un modèle relativement simple comparé à d'autres réseaux de classification d'images comme ImageNet [Krizhevsky et al., 2012] offre déjà des résultats intéressants (83% d'*accuracy* après 25 epochs pour notre modèle de 3 couches), et ce malgré une quantité de données limitée pour ce genre d'apprentissage. L'entraînement du réseau ImageNet a nécessité 1,3 millions de données alors que nous n'en avions qu'un peu plus de 15 000. Nos résultats sont similaires à ceux des différents participants au concours, qui présentaient un taux d'*accuracy* situé entre 80% et 90% pour les meilleurs [Stowell et al., 2016]. Cependant, les résultats du concours ont été calculés sur un jeu de données que nous n'avions pas à notre disposition. Par conséquent, nous ne pouvons pas comparer objectivement notre modèle à ceux des autres participants.

Les deux architectures de réseaux ne montrent pas de différence importante entre eux. Néanmoins, la suppression d'une partie des informations contenues dans les basses fréquences semble avoir un effet positif sur la précision de la prédiction pour les deux modèles. Le temps d'apprentissage n'a pas forcément besoin d'être long pour obtenir de bons résultats. Entre 25 à 35 epochs, la précision du modèle sur les données de validation se stabilise. Nous n'avons cependant pas laissé tourner les modèles assez longtemps que pour observer une stagnation de l'*accuracy* des données d'entraînement.

L'entraînement des réseaux de neurones est un processus de longue durée et les différentes configurations du réseau n'ont pas été testées un grand nombre de fois. Comme les écarts entre les différents résultats obtenus sont peu importants, nous ne pouvons affirmer avec certitude qu'ils sont principalement dus aux caractéristiques intrinsèques de ces modèles, plutôt qu'à l'usage de partitions différentes pour les données d'entraînement et de validation.

Nous estimons qu'avec une bibliothèque de sons plus fournie et un prétraitement du signal sonore plus efficace, des outils comme les réseaux de neurones permettraient d'améliorer nos capacités à observer les populations animales par des systèmes de détections automatisés renforçant les systèmes experts couramment utilisés.

Références

[Brownlee, 2017] Brownlee, J. (2017). Gentle introduction to the adam optimization algorithm for deep learning. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. Dernière consultation : 11-05-2020.

[Chollet, 2017] Chollet, F. (2017). Keras. https://keras.io/. Dernière consultation : 08-05-2020.

[Dahl et al., 2013] Dahl, G. E., Sainath, T. N., and Hinton, G. E. (2013). Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8609–8613. IEEE.

[Garg, 2019] Garg, R. (2019). https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks. Dernière consultation : 09-05-2020.

[Giacinto and Roli, 2001] Giacinto, G. and Roli, F. (2001). Design of effective neural network ensembles for image classification purposes. *Image and Vision Computing*, 19(9-10) :699–707.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

[Li et al., 2017] Li, J., Dai, W., Metze, F., Qu, S., and Das, S. (2017). A comparison of deep learning methods for environmental sound detection. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 126–130. IEEE.

[Litjens et al., 2016] Litjens, G., Sánchez, C. I., Timofeeva, N., Hermsen, M., Nagtegaal, I., Kovacs, I., Hulsbergen-Van De Kaa, C., Bult, P., Van Ginneken, B., and Van Der Laak, J. (2016). Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis. *Scientific reports*, 6 :26286.

[McFee et al., 2020] McFee, B., Lostanlen, V., McVicar, M., Metsai, A., Balke, S., Thomé, C., Raffel, C., Malek, A., Lee, D., Zalkow, F., Lee, K., Nieto, O., Mason, J., Ellis, D., Yamamoto, R., Seyfarth, S., Battenberg, E., , [U+FFFD], Bittner, R., Choi, K., Moore, J., Wei, Z., Hidaka, S., nullmightybofo, Friesch, P., Stöter, F.-R., Hereñú, D., Kim, T., Vollrath, M., and Weiss, A. (2020). librosa/librosa : 0.7.2. Dernière consultation : 13-05-2020.

[Nielsen, 2019] Nielsen, M. (2019). Improving the way neural networks learn. http://neuralnetworksanddeeplearning.com/chap3.html#the_cross-entropy_cost_function. Dernière consultation : 13-05-2020.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn : Machine learning in Python. *Journal of Machine Learning Research*, 12 :2825–2830.

[Piczak, 2015] Piczak, K. J. (2015). Environmental sound classification with convolutional neural networks. In *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE.

[Racine, 2019] Racine, L. (2019). Les fonctions d'activation. https://www.racinely.com/post/les-fonctions-d-activation-part-i. Dernière consultation : 11-05-2020.

[Ricco, 2017] Ricco, J. (2017). https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks. Dernière consultation : 09-05-2020.

[Simon, 2018] Simon, F. (2018). Deep Learning, les fonctions d'activation. https://www.supinfo.com/articles/single/7923-deep-learning-fonctions-activation. Dernière consultation : 11-05-2020.

[Smales, 2019] Smales, M. (2019). https://medium.com/@mikesmales/sound-classification-using-deep-learning-8bc2aa1990b7. Dernière consultation : 09-05-2020.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout : a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1) :1929–1958.

[Stowell et al., 2016] Stowell, D., Glotin, H., Stylianou, Y., and Wood, M. (2016). Bird audio detection challenge. http://machine-listening.eecs.qmul.ac.uk/bird-audio-detection-challenge/?fbclid=IwAR2IMmY_i8CMgUz3ldn5XLNGPhhS_62ZVims8H4KYQ53eM-6seJ9Z3566Xg#downloads. Dernière consultation : 08-05-2020.

[Stowell and Plumbley, 2013] Stowell, D. and Plumbley, M. D. (2013). An open dataset for research on audio field recording archives : freefield1010. https://arxiv.org/abs/1309.5275. Dernière consultation : 11-05-2020.

[Zhang, 2018] Zhang, C. (2018). Quick notes on how to choose optimizer in keras. https://www.dlology.com/blog/quick-notes-on-how-to-choose-optimizer-in-keras/. Dernière consultation : 11-05-2020.

5 Annexes

5.1 Fonction d'activation de type ReLU

La fonction rectified linear unit ou "ReLU" est une fonction d'activation linéaire par morceaux : elle remplace toute valeur d'entrée négative par 0 et toute valeur d'entrée positive par cette même valeur d'entrée. Elle est d'équation :

$$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases}$$

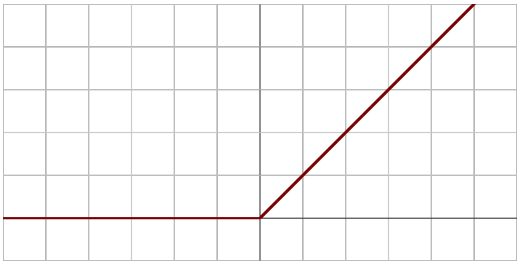

FIGURE 10 – Fonction ReLU [Racine, 2019]

Cette fonction est l'une des fonctions d'activation les plus utilisées car elle permet d'éviter notamment des problèmes de saturation [Simon, 2018].

5.2 Pooling

"Une couche de pooling est un processus de discrétisation basé sur des échantillons. L'objectif est de sous-échantillonner une représentation d'entrée (image, matrice de sortie de couche cachée, etc.), en réduisant sa dimensionnalité et en permettant de faire des hypothèses sur les caractéristiques contenues dans les sous-régions regroupées." [Ricco, 2017] Une fenêtre de max-pooling 2 x 2 consiste à résumer l'information contenue dans une sous-matrice de dimension 2 x 2 en ne gardant que la valeur maximale de la sous-matrice ainsi que l'illustre la Figure 11. Une fenêtre d'average pooling fonctionne d'une manière similaire mais résume l'information en prenant la moyenne de la sous-matrice plutôt que la valeur maximale.

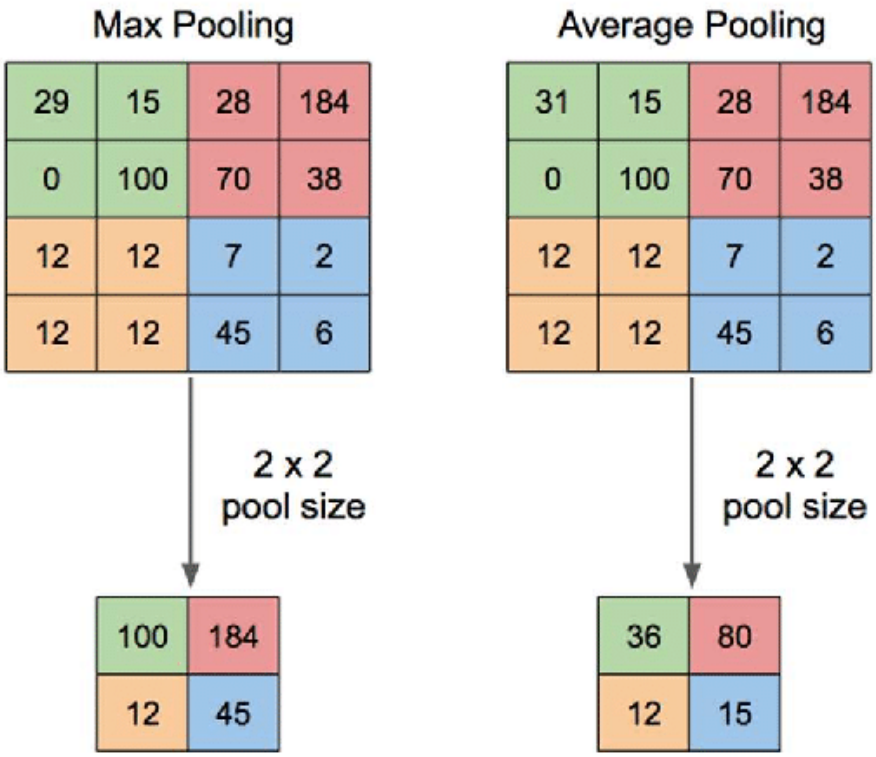

FIGURE 11 – Illustration du max pooling et de l'average pooling de fenêtre égal à 2 appliqué à une matrice [Garg, 2019]